WHAT IS CLAIMED IS:

1. A method for converting a C-type language program to a hardware design, comprising the steps of:

creating an algorithmic representation in a given C-type programming language corresponding to a preliminary hardware design; and

compiling the C-type programming language preliminary hardware design into a hardware description language (HDL) synthesizable design.

2. The method of claim 1, further comprising the step of synthesizing the HDL design into a gate-level hardware representation using a synthesis program to interpret the HDL design.

3. The method of claim 2, further comprising the step of using physical design tools to implement the gate-level representation as an actual hardware implementation.

4. The method of claim 1 wherein the C-type programming language is selected from among the group of C-type programming languages consisting of ANSI C, B, C++, Java, Kernighan & Ritchie C, and Objective C.

5. The method of claim 1 wherein the hardware description language is selected from among the group of hardware description languages consisting of Verilog, VHDL, ABEL, CUPL, AHDL, MACHX, and PALASM.

6. The method of claim 1, further comprising the step of simulating the HDL synthesizable design.

7. The method of claim 1 wherein the step of compiling comprises the steps of:

mapping predetermined C-type programming language expressions to functionally equivalent HDL program language expressions; and

assigning input/output as defined in the C-type program to specific wires in the HDL synthesizable design; and further comprising the step of:

configuring in the HDL synthesizable design an interface for the gate-level hardware representation.

8. The method of claim 7 wherein a plurality of selected C-type functions that can execute simultaneously are compiled into a plurality of executable HDL program language expressions that operate in parallel.

9. The method of claim 7 wherein a plurality of interdependent C-type functions are compiled into a plurality of executable HDL program language expressions that operate one of simultaneously and sequentially in a data processing pipeline.

10. The method of claim 7 wherein a callable C-type function is compiled into an executable HDL program language expression that is executed one of synchronously and asynchronously on the occurrence of an external event.

11. The method of claim 7 wherein the C-type programming language expressions include at least one of integer, floating point, pointer, enum, struct, and union variables.

12. The method of claim 7, further comprising the step of compiling C-type programming language structure assignment, structure function parameters, and structure function return values into HDL synthesizable expressions.

13. The method of claim 1 wherein the step of compiling comprises the steps of:

compiling a C-type program control flow into an HDL state machine; and

assigning input/output as defined in the C-type program to specific wires in the HDL synthesizable design; and further comprising the step of:

configuring in the HDL synthesizable design an interface for the gate-level hardware representation.

14. The method of claim 13, further comprising the steps of:

determining the presence of any C-type pointers in the C-type program; and

compiling any C-type pointers and pointer indirection into an HDL state-machine-based memory access protocol.

15. The method of claim 13, further comprising the steps of:

determining the presence of any non-addressable variables in the C-type program; and

mapping any non-addressable variables onto hardware registers formed in HDL.

16. The method of claim 13, further comprising the steps of:

determining the presence of any addressable variables in the C-type program; and

mapping any addressable variables onto at least one of addressable hardware memory and an addressable array of hardware registers formed in HDL.

17. The method of claim 13, further comprising the steps of:

determining the presence of any complicated C-type mathematical operations in the C-type program; and

compiling multiple occurrences of any complicated C-type mathematical operation into a given HDL functional block that implements such an operation invoked on each occurrence by different states driven by the state machine.

18. The method of claim 13 wherein a plurality of selected C-type functions that can execute simultaneously are compiled into a plurality of HDL state machines that operate in parallel.

19. The method of claim 13 wherein a plurality of interdependent C-type functions are compiled into a plurality of executable HDL program language expressions that operate one of simultaneously and sequentially in a data processing pipeline.

20. The method of claim 13 wherein a callable C-type function is compiled into at least one executable HDL state machine which is executed one of synchronously and asynchronously on the occurrence of an external event.

21. The method of claim 15 wherein the C-type programming language variables include at least one of integer, floating point, pointer, enum, struct, and union variables.

22. The method of claim 16 wherein the C-type programming language variables include at least one of integer, floating point, pointer, enum, struct, and union variables.

23. The method of claim 13, further comprising the step of compiling C-type programming language structure assignment, structure function parameters, and structure function return values into HDL synthesizable expressions.

24. The method of claim 13, further comprising the step of determining the presence of any recursion in the C-type program, and wherein at least one of an identifiable direct and indirect recursive function call is compiled into an HDL state machine which has an interface to a stack implemented utilizing one of external and internal memory and an array of hardware registers, wherein the state machine has a stack pointer implemented utilizing a hardware register employed to store and restore local variables of the recursive function and other recursive function information in the stack.

25. A method for converting a high-level language program to a hardware design, comprising the steps of:

creating an algorithmic representation in a given high-level programming language corresponding to a preliminary hardware design;

translating the high-level programming language preliminary hardware design into a C-type programming language preliminary hardware design; and

compiling the C-type programming language preliminary hardware design into a hardware description language (HDL) synthesizable design.

26. The method of claim 25, further comprising the step of synthesizing the HDL design into a gate-level hardware representation using a synthesis program to interpret the HDL design.

27. The method of claim 26, further comprising the step of using physical design tools to implement the gate-level hardware representation as an actual hardware implementation.

28. The method of claim 25 wherein the high-level programming language is selected from among the group of C-type programming languages consisting of APL, Ada, Algol, B, Basic, Kernighan & Ritchie C, C++, CLOS, COBOL, Clu, Common Lisp, Coral, Dylan, Eiffel, Emacs Lisp, Forth, Fortran, IDL, Icon, Java, Jovial, Lisp, LOGO, ML, Modula, Oberon, Objective C, PL/I, PL/M, Pascal, Postscript, Prolog, Python, RTL, Rexx, SETL, Simula, Sather, Scheme, Smalltalk, Standard ML, TCL, and TRAC.

29. The method of claim 25 wherein the hardware description language is selected from among the group of hardware description languages consisting of Verilog, VHDL, ABEL, CUPL, AHDL, MACHX, and PALASM.

30. The method of claim 25, further comprising the step of simulating the HDL synthesizable design.

31. The method of claim 25 wherein the step of compiling comprises the steps of:

mapping predetermined C-type programming language expressions to functionally equivalent HDL program language expressions; and

assigning input/output as defined in the C-type program to specific wires in the HDL synthesizable design; and further comprising the step of:

configuring in the HDL synthesizable design an interface for the gate-level hardware representation.

32. The method of claim 31 wherein a plurality of selected C-type functions that can execute simultaneously are compiled into a plurality of executable HDL program language expressions that operate in parallel.

33. The method of claim 31 wherein a plurality of interdependent C-type functions are compiled into a plurality of executable HDL program language expressions that operate one of simultaneously and sequentially in a data processing pipeline.

34. The method of claim 31 wherein a callable C-type function is compiled into an executable HDL program language expression that is executed one of synchronously and asynchronously on the occurrence of an external event.

35. The method of claim 31 wherein the C-type programming language expressions include at least one of integer, floating point, pointer, enum, struct, and union variables.

36. The method of claim 31, further comprising the step of compiling C-type programming language structure assignment, structure function parameters, and structure function return values into HDL synthesizable expressions.

37. The method of claim 25 wherein the step of compiling comprises the steps of:

compiling a C-type program control flow into an HDL state machine; and

assigning input/output as defined in the C-type program to specific wires in the HDL synthesizable design; and further comprising the step of:

configuring in the HDL synthesizable design an interface for the gate-level hardware representation.

38. The method of claim 37, further comprising the steps of:

determining the presence of any C-type pointers in the C-type program; and

compiling any C-type pointers and pointer indirection into an HDL state-machine-based memory access protocol.

39. The method of claim 37, further comprising the steps of:

determining the presence of any non-addressable variables in the C-type program; and

mapping any non-addressable variables onto hardware registers formed in HDL.

40. The method of claim 37, further comprising the steps of:

determining the presence of any addressable variables in the C-type program; and

mapping any addressable variables onto at least one of addressable hardware memory and an addressable array of hardware registers formed in HDL.

41. The method of claim 37, further comprising the steps of:

determining the presence of any complicated C-type mathematical operations in the C-type program; and

compiling multiple occurrences of any complicated C-type mathematical operation into a given HDL functional block that implements such an operation invoked on each occurrence by different states driven by the state machine.

42. The method of claim 37 wherein a plurality of selected C-type functions that can execute simultaneously are compiled into a plurality of HDL state machines that operate in parallel.

43. The method of claim 37 wherein a plurality of interdependent C-type functions are compiled into a plurality of executable HDL program language expressions that operate one of simultaneously and sequentially in a data processing pipeline.

44. The method of claim 37 wherein a callable C-type function is compiled into at least one executable HDL state machine which is executed one of synchronously and asynchronously on the occurrence of an external event.

45. The method of claim 39 wherein the C-type programming language variables include at least one of integer, floating point, pointer, enum, struct, and union variables.

46. The method of claim 40 wherein the C-type programming language variables include at least one of integer, floating point, pointer, enum, struct, and union variables.

47. The method of claim 37, further comprising the step of compiling C-type programming language structure assignment, structure function parameters, and structure function return values into HDL synthesizable expressions.

48. The method of claim 37, further comprising the step of determining the presence of any recursion in the C-type program, and wherein at least one of an identifiable direct and indirect recursive function call is compiled into an HDL state machine which has an interface to a stack implemented utilizing one of external and internal memory and an array of hardware registers, wherein the state machine has a stack pointer implemented utilizing a hardware register employed to store and restore local variables of the recursive function and other recursive function information in the stack.

49. A method for converting a high-level algorithmic programming language program to a hardware design description, comprising the steps of:

selecting a given high-level algorithmic programming language having at least one of pointers and structures;

creating an algorithmic representation in the given high-level algorithmic programming language corresponding to a preliminary hardware design; and

compiling the high-level programming language preliminary hardware design into hardware-specific descriptions for a hardware implementation.

50. The method of claim 49 wherein the step of compiling comprises the step of compiling the high-level algorithmic programming language preliminary hardware design into a hardware design language (HDL) synthesizable design.

51. The method of claim 50, further comprising the step of synthesizing the HDL design into a gate-level hardware representation using a synthesis program to interpret the HDL design.

52. The method of claim 51, further comprising the step of using physical design tools to implement the gate-level representation as an actual hardware implementation.

53. The method of claim 49 wherein the high-level algorithmic programming language is selected from among the group of C-type programming languages consisting

of APL, Ada, Algol, ANSI C, B, Basic, Kernighan & Ritchie C, C++, CLOS, COBOL, Clu, Common Lisp, Coral, Dylan, Eiffel, Emacs Lisp, Forth, Fortran, IDL, Icon, Java, Jovial, Lisp, LOGO, ML, Modula, Oberon, Objective C, PL/I, PL/M, Pascal, Postscript, Prolog, Python, RTL, Rexx, SETL, Simula, Sather, Scheme, Smalltalk, Standard ML, TCL, and TRAC.

54. The method of claim 50 wherein the hardware design language is selected from among the group of hardware design languages consisting of Verilog, VHDL, ABEL, CUPL, AHDL, MACHX, and PALASM.

55. The method of claim 50, further comprising the step of simulating the HDL synthesizable design.